Sid Chatterjee, Manish Gupta, José E. Moreira

Case study: Parallel LU factorization on BG/L

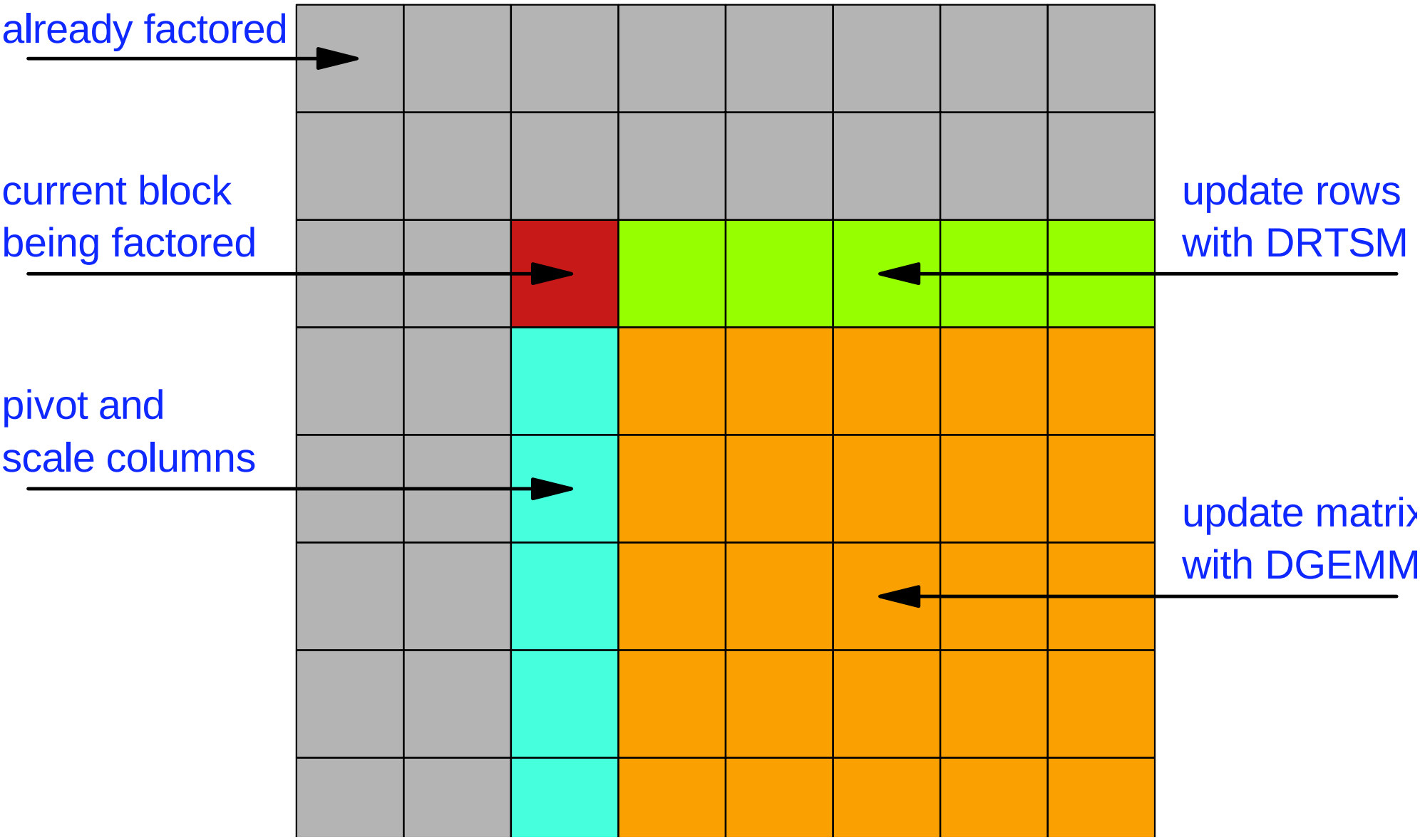
or

how I stopped worrying and learned to love BG/L

Making good use of BG/L with MPI

- IBM's BlueGene/L is a feature-rich architecture:
  - Torus interconnect for high-bandwidth communication
  - Multicast operations along any axis of the torus
  - Tree interconnect for low-latency/high-bandwidth reduces/broadcasts
  - 16-byte floating-point unit capable of 4 floating-point operations per cycle
  - Two processors/node
- Exploiting those features requires several levels of support
  - Compilers must be able to extract and expose instruction-level parallelism
  - Libraries must be coded so that they use the features
  - Application programs must be coded so that features can be used
- This talk is about organizing your MPI program so that it can take advantage of the "cool" features of BlueGene/L
- It is organized as a case study of a well-know computation: Blocked LU factorization

Blocked LU factorization - an algorithm



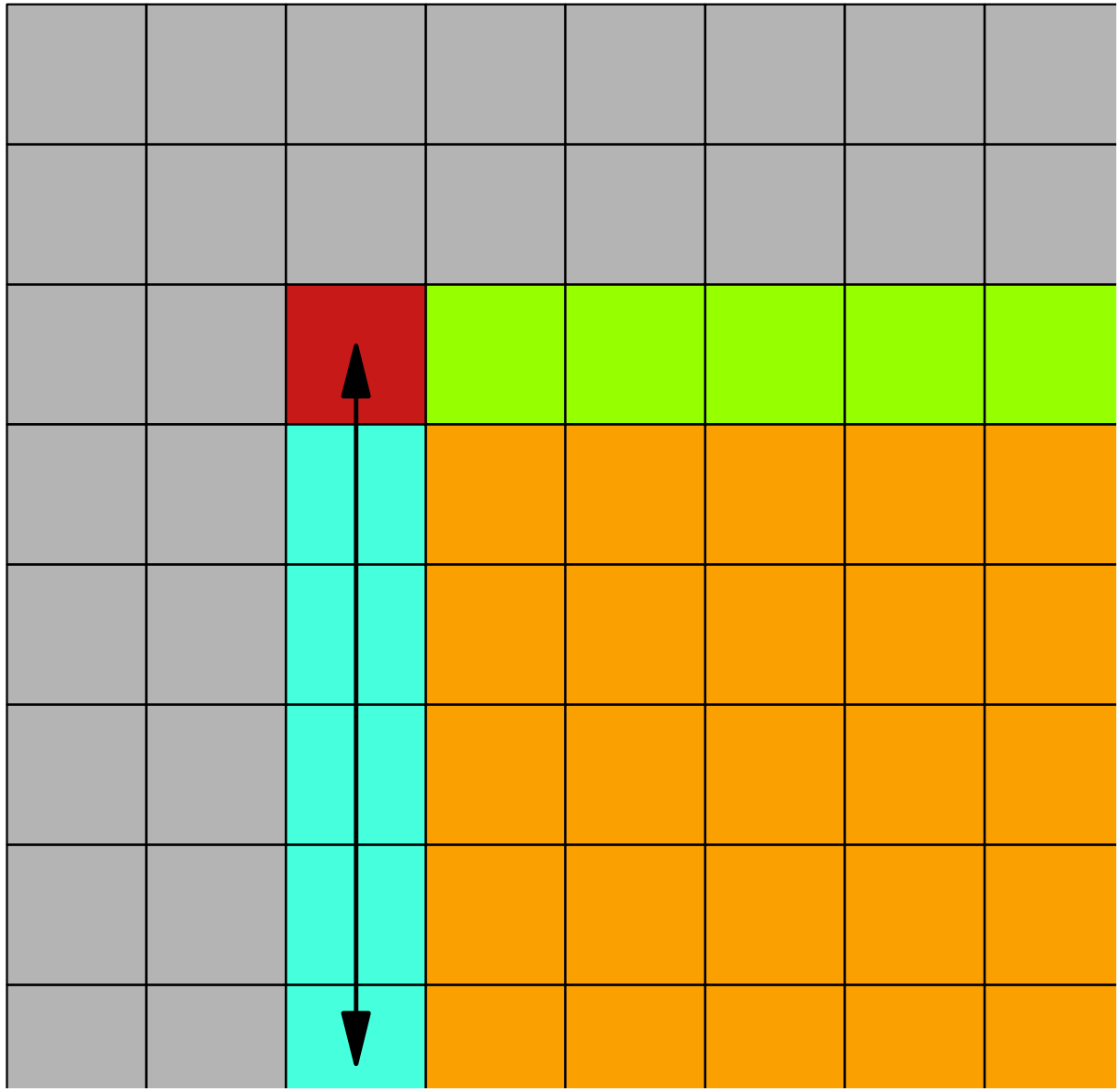
Blocked LU factorization - step 1

Start with next diagonal block

Pivot and update columns

- find maximum absolute value
- exchange with diagonal
- update next columns

At the end, red block is in LU factored form, blue blocks are updated

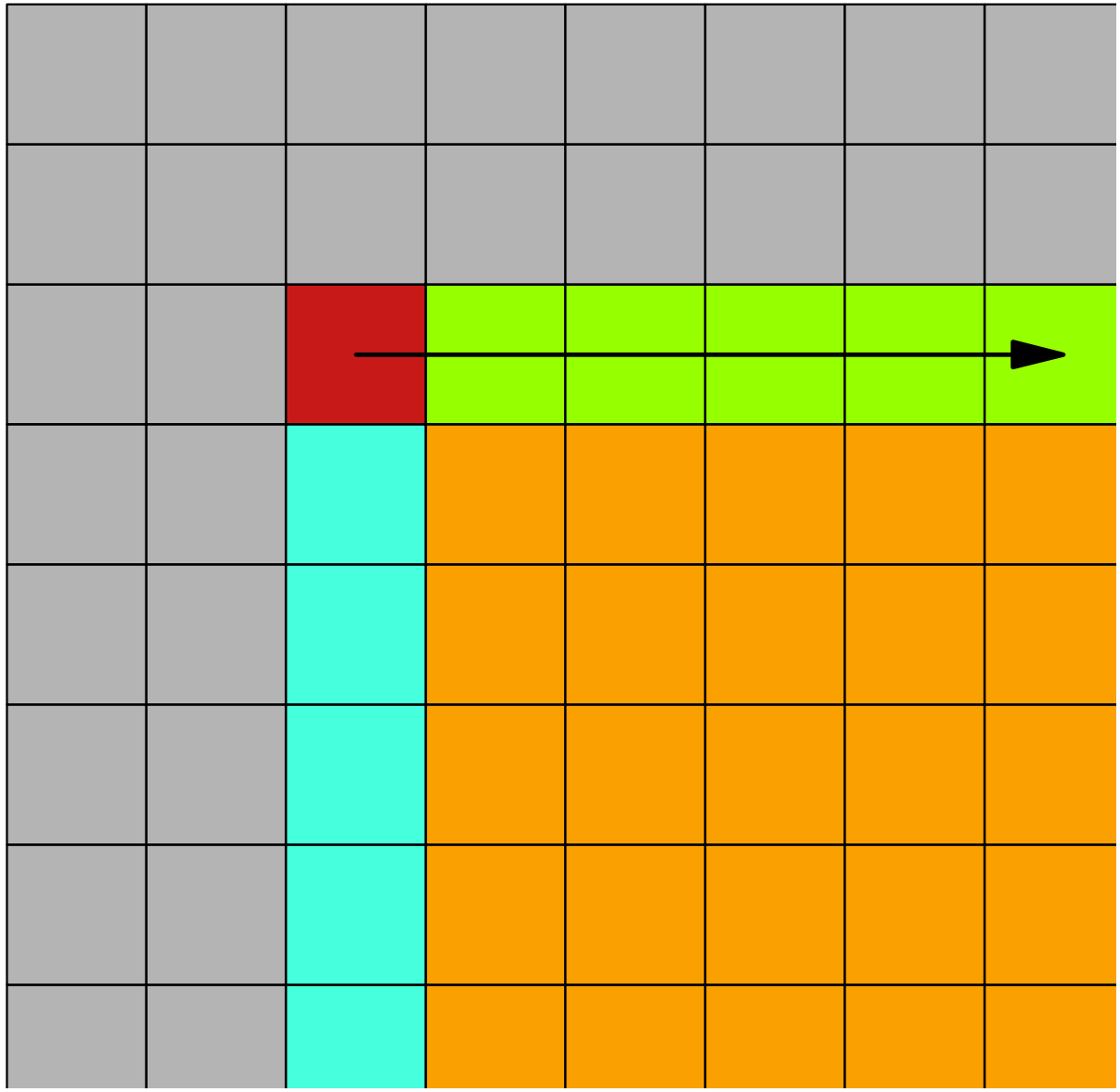


Blocked LU factorization - step 2

Update row of green blocks

- use lower factor of red block with each green block
- DTRSM operation

At the end, green blocks are updated



Blocked LU factorization - step 3

Update each orange block

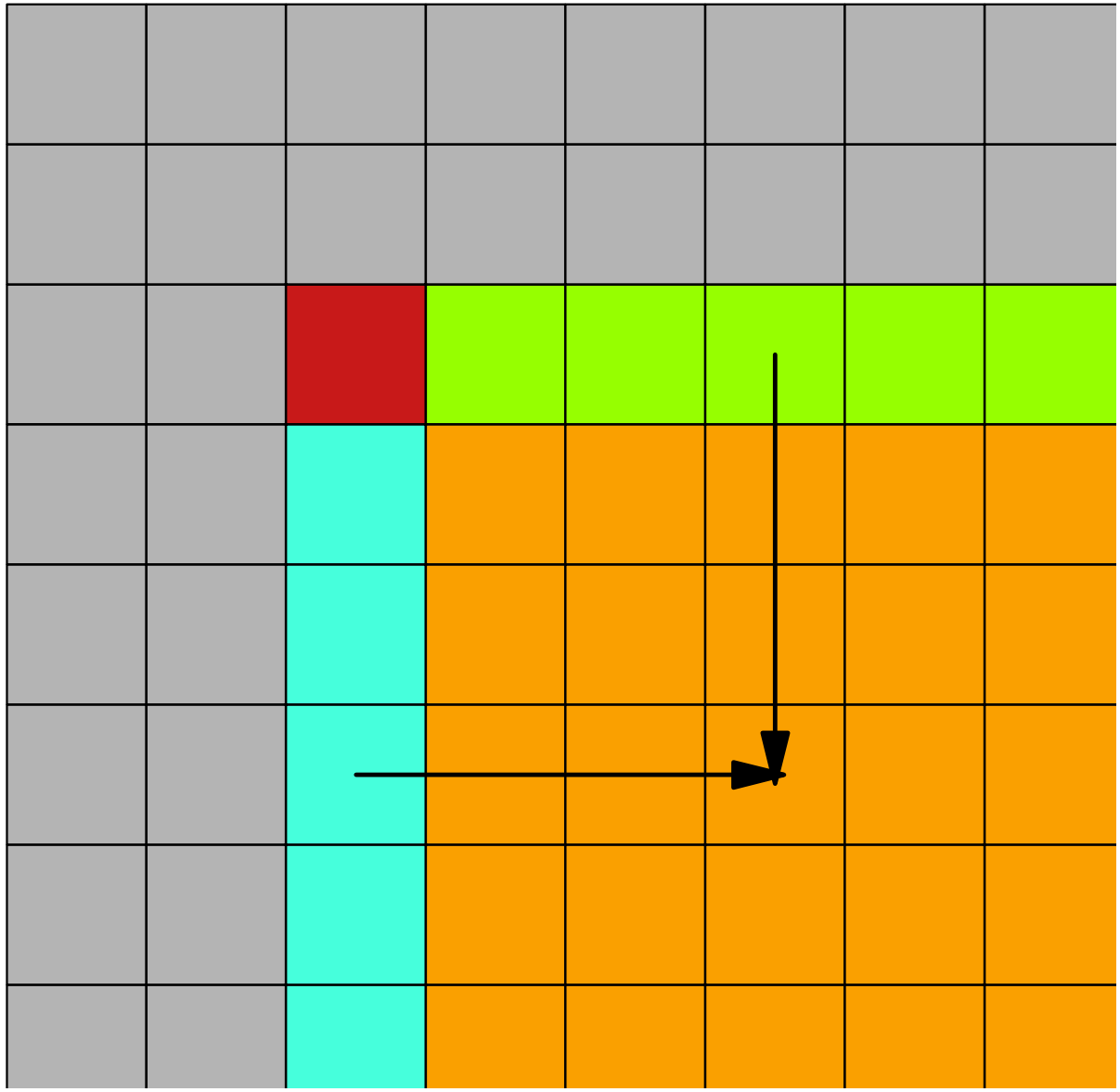
- use green block of column and blue block of row
- DGEMM operation

Each blue block is used in every orange block of row

Each green block is used in every orange block of column

At the end, orange blocks are updated

LU factorization continues with next diagonal block



Parallel LU factorization

We partition the matrix as an 8x8 array of blocks

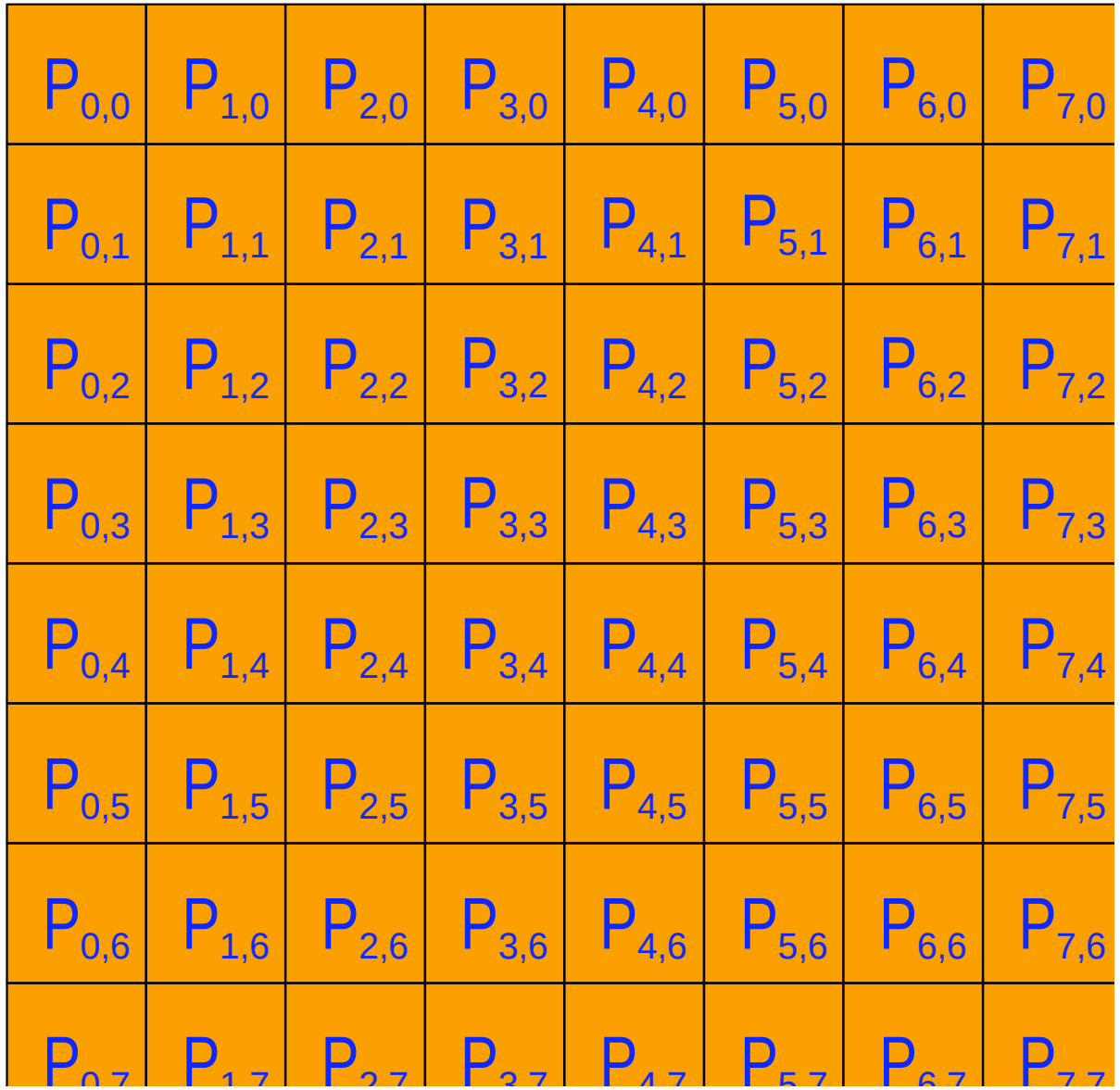
We execute the factorization on an 8x8 logical grid of processes

The data is decomposed using a block distribution

(In practice, block-cyclic distribution gives better utilization)

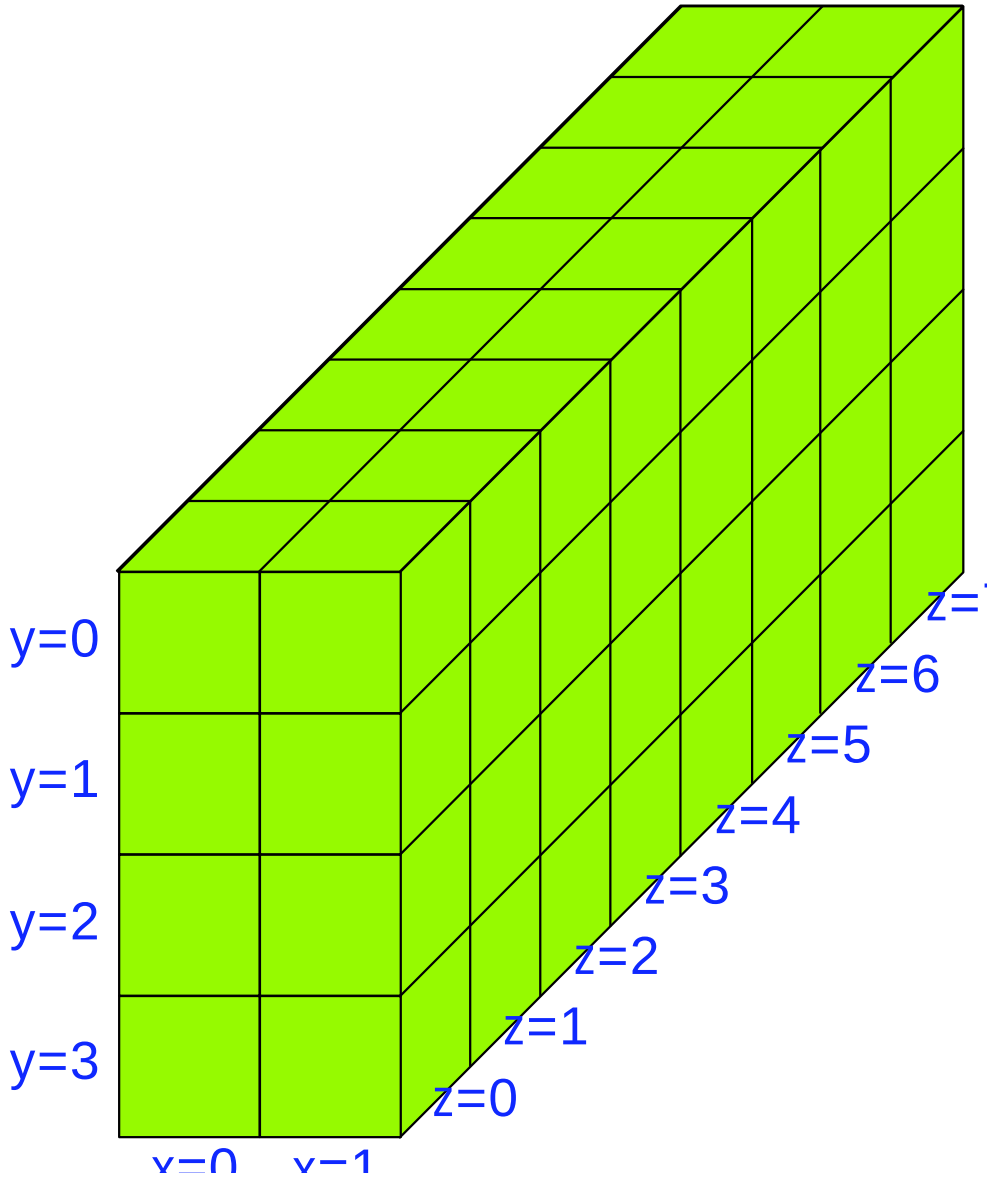
To run on BG/L:

- request the appropriate number of compute nodes
- create a logical two-dimensional grid of processes
- perform operations on that logical grid



Creating a physical node partition

- Node partitions are created when jobs are scheduled for execution
- User specifies desired shape when submitting job:
  - **submit lufact 2x4x8**
  - request a job partition of 64 compute nodes, with shape 2 (on x-axis) by 4 (on y-axis) by 8 (on z-axis)
- A contiguous, rectangular subsection of the compute nodes is carved out for this job
- Nodes are indexed by their (x,y,z) coordinates inside the job partition



Mapping processes to physical nodes

In MPI, logical process grids are created with **MPI_CART_CREATE**

The mapping is performed by the system, matching physical topology

In this case, we have mapped each xy-plane to one column

Within a column, consecutive values of y are neighbors

Logical row operations correspond to operations on a string of physical nodes along the z-axis

Logical column operations correspond to operations on an xy-plane

row and column communicators are created with **MPI_CART_SUB**

0,0,0	0,0,1	0,0,2	0,0,3	0,0,4	0,0,5	0,0,6	0,0,7
0,1,0	0,1,1	0,1,2	0,1,3	0,1,4	0,1,5	0,1,6	0,1,7
0,2,0	0,2,1	0,2,2	0,2,3	0,2,4	0,2,5	0,2,6	0,2,7
0,3,0	0,3,1	0,3,2	0,3,3	0,3,4	0,3,5	0,3,6	0,3,7
1,0,0	1,0,1	1,0,2	1,0,3	1,0,4	1,0,5	1,0,6	1,0,7
1,1,0	1,1,1	1,1,2	1,1,3	1,1,4	1,1,5	1,1,6	1,1,7
1,2,0	1,2,1	1,2,2	1,2,3	1,2,4	1,2,5	1,2,6	1,2,7
1,3,0	1,3,1	1,3,2	1,3,3	1,3,4	1,3,5	1,3,6	1,3,7

Creating communicators

First, create a two-dimensional 8x8 cartesian communicator:

GRID2D_COMM = MPI_CART_CREATE(MPI_COMM_WORLD, 2, 8x8)

Then, create a communicator along the row of each process:

ROW_COMM = MPI_CART_SUB(GRID2D_COMM, [true, false)

Finally, create a communicator along the column of each process:

COL_COMM = MPI_CART_SUB(GRID2D_COMM, [false, true])

Blocked LU factorization - step 1

Pivot and update columns

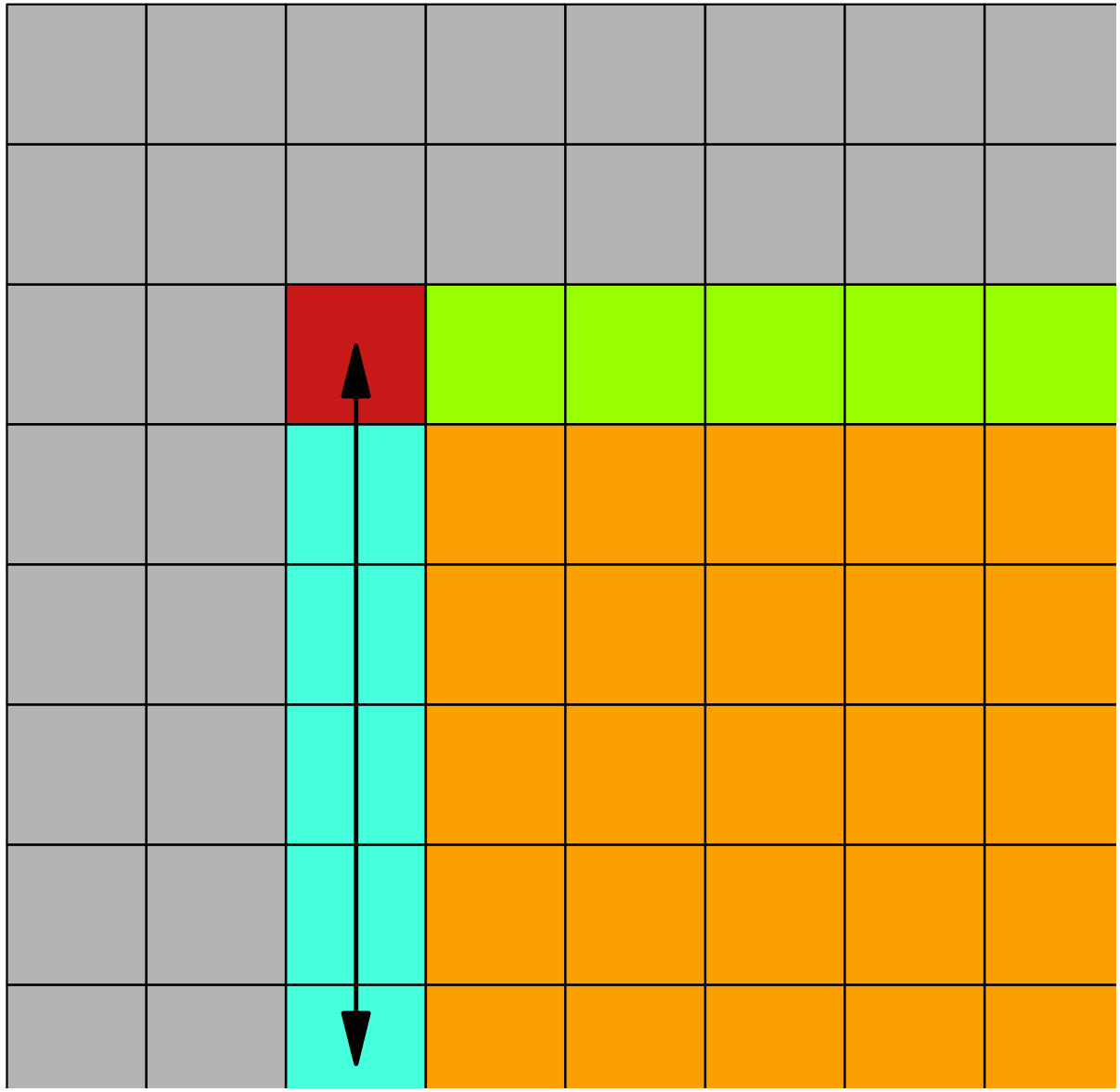
- find maximum absolute value
- exchange with diagonal
- update next columns

At the end, red block is in LU factored form, blue blocks are updated

Each process finds its maximum absolute value in column

MPI_REDUCE is used to find column maximum

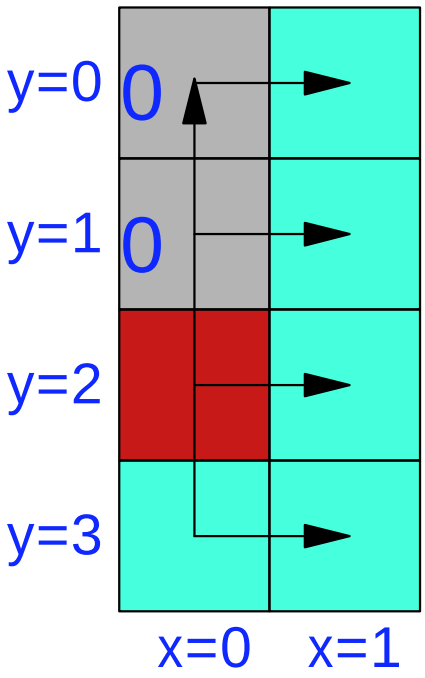
- operation on a logical column of processes = xy-plane in physical partition



Computing global maximum

- Expressing operation in MPI:
  pivot = MPI_REDUCE(local_max, MAX, COL_COMM)

- Performing the operation on a physical plane:



- One option is to map the reduction to the tree:
  - use different classes for subsets of processes
  - in general, cannot be done for all desired subsets
- Another option is to use row multicast to flood the plane with local maxima
  - takes advantages of high bandwidth in torus
- Or, can just use point-to-point communications to perform reduction as data is moved

Blocked LU factorization - step 2

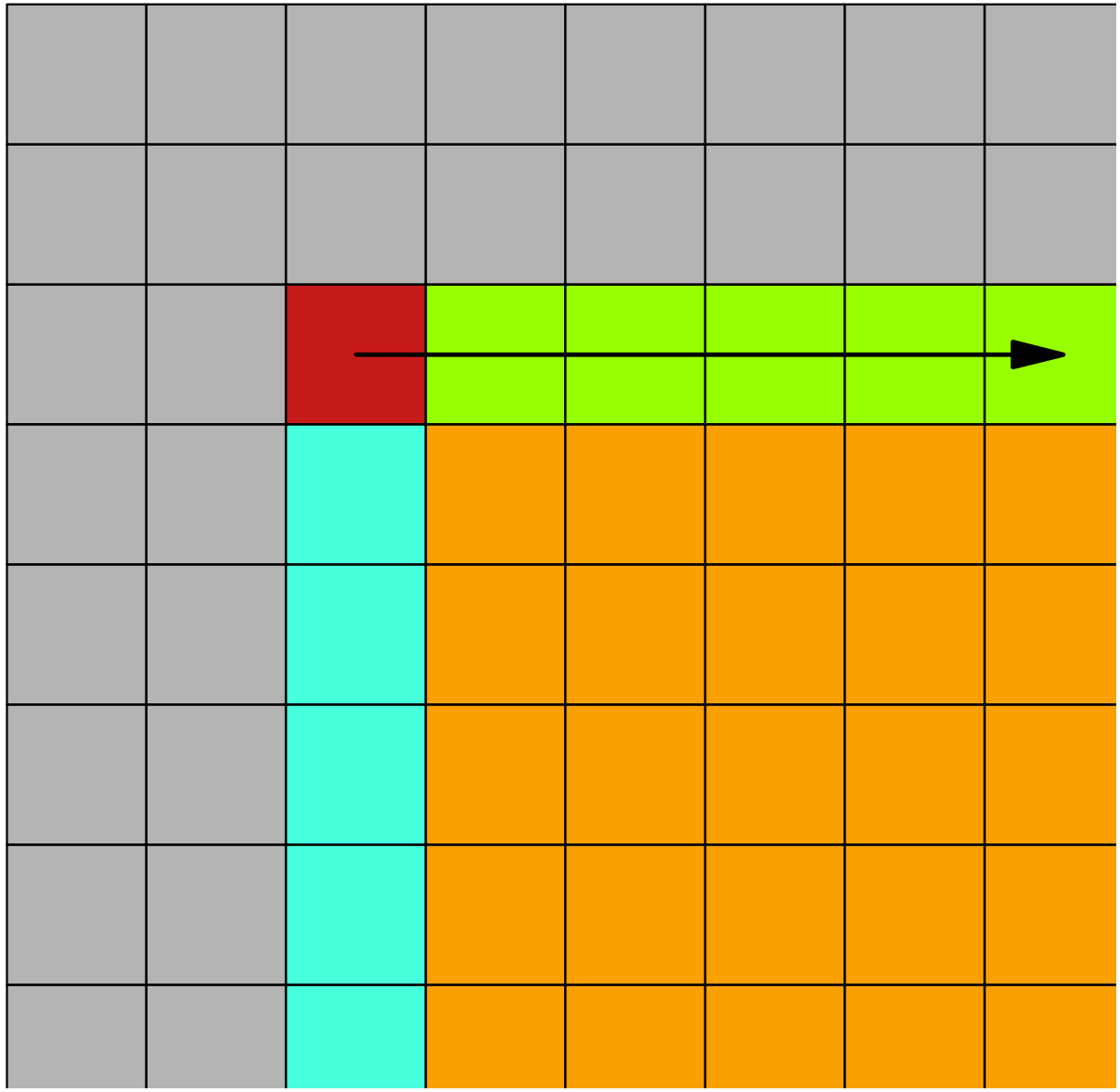
Update row of green blocks

- use lower factor of red block with each green block
- DTRSM operation

At the end, green blocks are updated

MPI_BCAST is used to distribute factor:

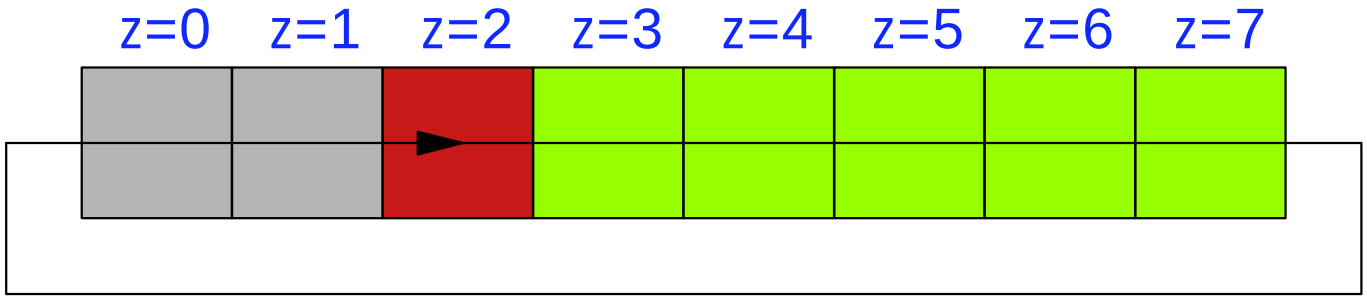
- operation on logical row of processes = z-axis of physical partition



Broadcasting along logical row

- Expressing operation in MPI:
  MPI_BCAST(block, ROW_COMM)

- Performing operation on a physical string of nodes:



- A simple multicast to all nodes in z-axis will do the trick

Blocked LU factorization - step 3

Update each orange block

- use green block of column and blue block of row
- DGEMM operation

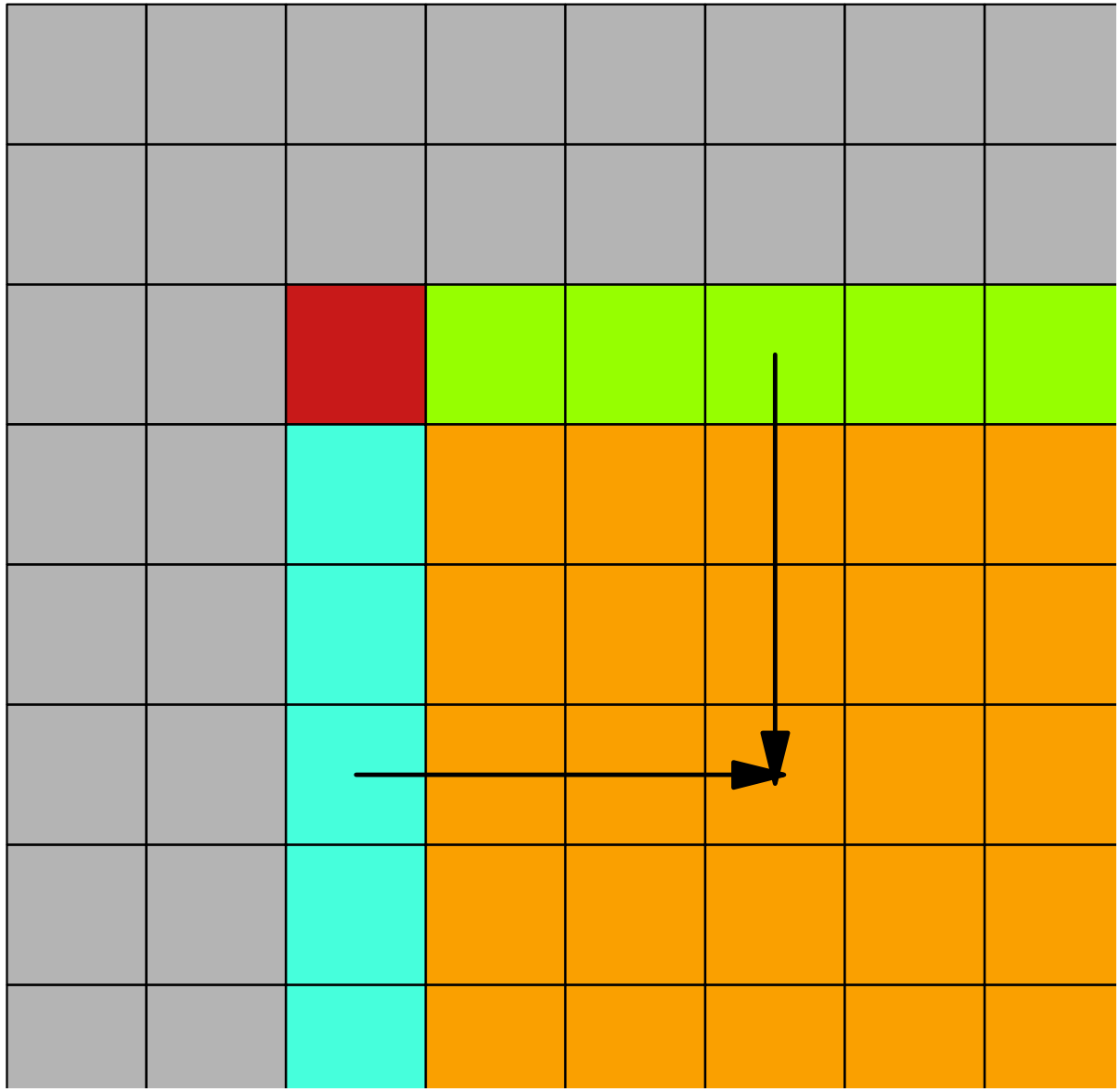
Each blue block is used in every orange block of row

Each green block is used in every orange block of column

At the end, orange blocks are updated

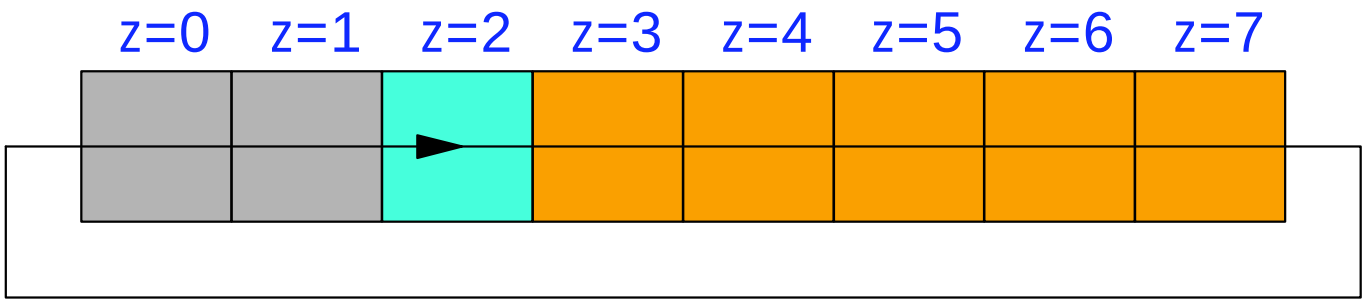
LU factorization continues with next diagonal block

MPI_BCAST is used to distribute blue blocks along row and green blocks along columns

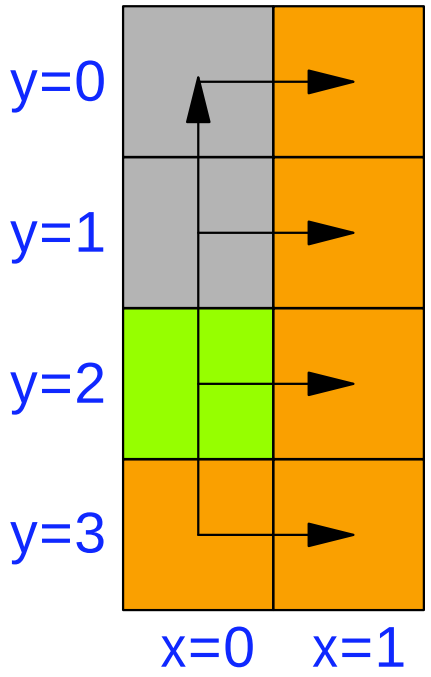


Broadcasting along logical rows / columns

- Expressing operation in MPI:
  MPI_BCAST(block, ROW_COMM)
  MPI_BCAST(block, COL_COMM)
- Performing operation on a physical string of nodes alon z-axis:



  - A simple multicast to all nodes in z-axis will do the trick for a blue block
- Performing operation on a physical xy-plane:
  - A double multicast, first along y-axis and then along x-axis will flood the plane with a green block



Conclusions

- Efficient mapping of operations on logical grid to real compute nodes require some forethought: interaction between job submission and run-time
- Broadcasts and reduces can be performed very efficiently in BG/L if they map to regular subsections of compute node grid
  - strings along an axis
  - planes of the three-dimensional interconnect
- We did not mention in this talk, but "where" the data is in memory is very important to BG/L
  - performance critical data should be 16-byte aligned
  - requirement for both send and received
- Goal is to simplify life of programmer, let the system do it!